1

TEXTO BASE

PROGRAMAÇÃO ORIENTADA A OBJETOS

Faculdade IMPACTA

Texto base

1

Apresentação da disciplina e preparação do ambiente de desenvolvimento

Prof. MSc. Rafael Maximo Carreira Ribeiro

Resumo

Neste capítulo vamos configurar o ambiente de desenvolvimento que usaremos na disciplina, discutir brevemente os princípios básicos que nortearam e ainda norteiam o desenvolvimento da linguagem Python e introduzir o guia de estilo elaborado pela comunidade para melhorar a legibilidade e consistência de códigos escritos em Python ao redor de todo o mundo.

1.1. Agradecimentos

Agradeço aos Professores Me. Fernando Sousa, Me. Leonardo Takuno, Me. Renan Rodrigues, Dr. Renato Rodrigues Oliveira da Silva e Esp. Vilson Ferreira que contribuíram com a elaboração da primeira apostila do curso e que serviu de base para guiar o desenvolvimento da atual versão.

1.2. Introdução

A disciplina de Programação Orientada a Objetos é uma continuação do que foi visto em Linguagem de Programação (LP). Nesta disciplina vocês conhecerão mais estruturas de programação utilizadas em Python, e que podem também ser aplicadas em outras linguagens de programação, para poder resolver problemas ainda mais complexos.

Tudo que você aprendeu em LP será aproveitado aqui: variáveis, tipos de dados, estruturas de decisão, estruturas de repetição, listas, tuplas e funções. Isso é na realidade a base para todas as disciplinas que envolvem programação no curso e não será diferente para esta. Utilizaremos tais conceitos para aprender um novo paradigma de desenvolvimento de software, no qual abordamos a representação dos problemas baseada na construção de objetos e na interação entre eles, traçando um paralelo mais próximo entre a realidade e a programação.

Esse paradigma é conhecido como Programação Orientada a Objetos, que chamaremos de POO de agora em diante. Aprenderemos o que são objetos, classes, métodos e atributos, veremos qual a relação entre eles e conceitos como abstração, encapsulamento, herança e polimorfismo, que formam os pilares de POO.

Além de POO, aprenderemos também sobre como criar módulos em Python, estudaremos como podemos testar nossas aplicações de maneira automatizada e veremos uma introdução aos princípios do SOLID e aos padrões de projeto. Estes fatores contribuem para um código de manutenção mais fácil, com menos erros ou *bugs* e portanto com maior qualidade.

Esta disciplina é fundamental para sua formação na área de tecnologia da informação, mesmo que você não venha a trabalhar diretamente como desenvolvedor, os conceitos aprendidos aqui com certeza serão um diferencial na sua carreira.

VOCÊ SABIA?

A linguagem Python foi criada há mais de 30 anos, e a primeira implementação foi feita em um Mac, em 1989, pelo seu criador Guido van Rossum. O nome é uma homenagem à série de TV britânica *Monty Python's Flying Circus*, do começo da década de 1970. Guido criou a versão zero da linguagem como um hobby, durante um feriado de natal, com o intuito de criar uma linguagem fácil de aprender e cujo código seja agradável de ler, e um dos principais recursos que contribuiu para isso foi a decisão de definir os blocos de código através da indentação. (Rossum, G. V., 1996).

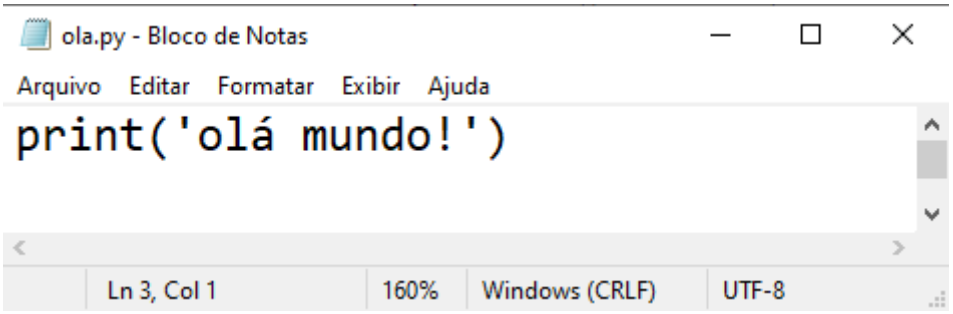
1.3. Configuração do ambiente de desenvolvimento

A disciplina de Programação Orientada a Objetos utiliza como base a linguagem de programação Python, dando sequência ao que foi visto em LP. Você precisará ter instalado em seu computador a versão 3 do Python, de preferência a mais recente, mas para esta disciplina qualquer versão igual ou superior a 3.7 será suficiente. Se precisar, baixe a versão mais recente em https://www.python.org/downloads/. Este é o mesmo instalador utilizado na disciplina de LP, e a configuração de instalação será exatamente a mesma. O que muda agora é o editor que utilizaremos para escrever os programas.

Um programa de computador ou um projeto pode ser desenvolvido utilizando apenas o bloco de notas. Um arquivo de código-fonte em Python nada mais é que um arquivo de texto codificado utilizando a tabela UTF-8[1] e com a extensão *.py ao invés de *.txt. O programa da Figura 1.1 foi feito no bloco de notas, e podemos abrir um *prompt* de comando no windows e executá-lo diretamente, como mostra a Figura 1.2, o único requisito é possuir o interpretador do Python instalado e acessível.

[1] Observe a codificação (*encoding*) usada pelo bloco de notas no canto inferior direito da figura, caso o seu editor use uma codificação diferente, você deve alterá-la antes de salvar o arquivo. Atualmente, a maioria dos editores de texto já utilizam UTF-8 por padrão.

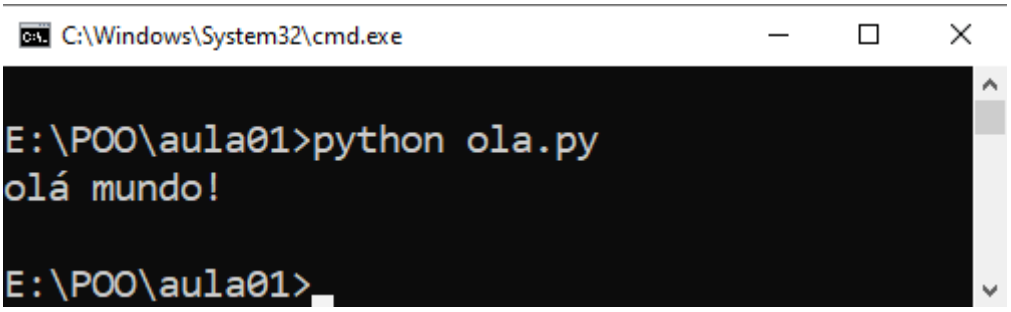
Figura 1.1: Programa “ola.py” escrito usando o bloco de notas



Fonte: do autor, 2021

Este mesmo teste pode igualmente ser realizado em outros sistemas operacionais como Linux e MacOS, com qualquer editor de texto disponível nestas plataformas e usando o terminal para executar o arquivo.

Figura 1.2: Execução do arquivo “ola.py” no promt de comando do Windows

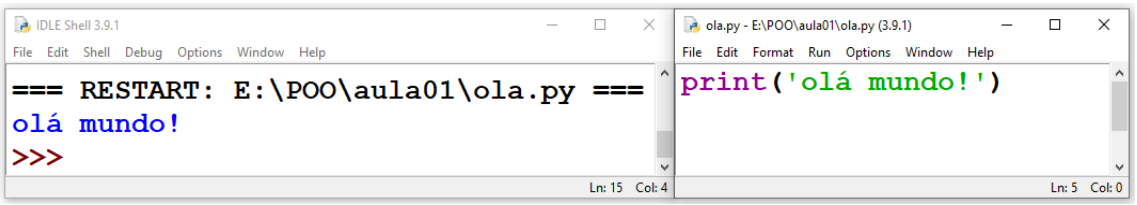


Fonte: do autor, 2021

A linguagem Python possui o próprio Ambiente de Desenvolvimento Integrado ou IDE, da sigla em inglês *Integrated Development Environment*, chamado IDLE. A IDLE é uma IDE voltada para o aprendizado da linguagem Python, o L vem de *Learning*, e é instalado automaticamente junto com a instalação do interpretador do Python (nos sistemas Windows, para Linux pode ser necessária a instalação à parte da IDLE).

A IDLE é um programa que contém um console interativo para executar comandos do Python, chamado *Shell*, e um editor de texto próprio, com realce de sintaxe e outras ferramentas simples. Veja um exemplo na Figura 1.3 a seguir.

Figura 1.3: Código fonte Python aberto no editor do IDLE (à direita) e executado na *Shell* do Python (à esquerda)



Fonte: do autor, 2021

Uma IDE, de forma geral, é um software que provê facilidade quando estamos escrevendo um programa, com o intuito de aumentar a produtividade ao integrar diversas ferramentas em um mesmo ambiente. Como vimos, a IDLE provê a coloração do código (realce de sintaxe), que facilita a identificação de comandos e estruturas, e a integração com a *Shell*, que nos permite executar o programa escrito e inspecionar as variáveis criadas após sua execução.

A definição do que um programa precisa para ser considerado uma IDE pode variar, mas deve incluir pelo menos:

- Um editor de texto para a linguagem em questão;
- Possibilidade de gerenciamento de arquivos;
- Construção automática, isto é, um compilador ou interpretador capaz de gerar o código binário e realizar os processos necessários para que o código possa ser executado;
- Uma ferramenta de depuração (debugger).

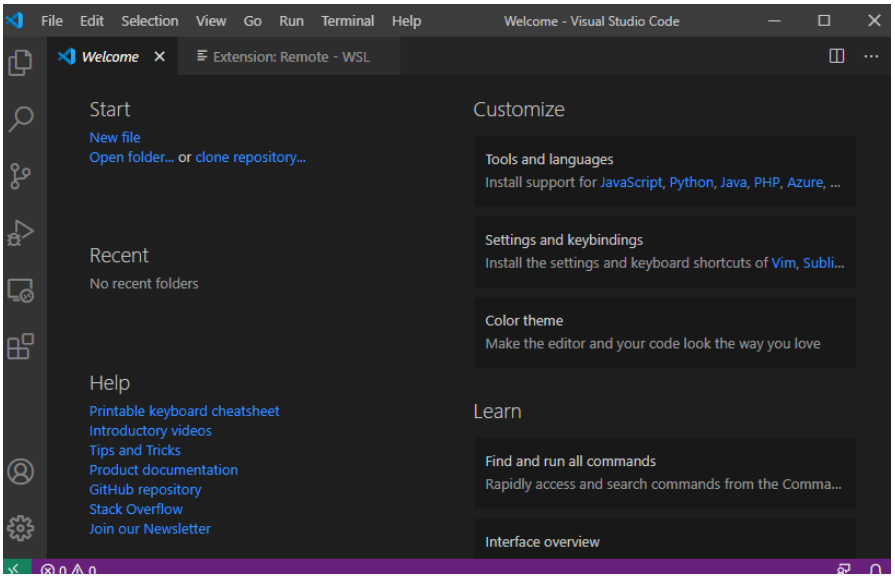
Outras funcionalidades muito comuns em IDEs modernas incluem:

- Função de autocompletar os comandos digitados;
- Integração com ferramentas para, entre outros:
    - Versionamento (GIT);
    - Construção de interfaces gráficas (GUI); e
    - Construção de diagramas.

Neste curso utilizaremos o Visual Studio Code, que abreviaremos para VSCode, uma IDE moderna desenvolvida pela Microsoft, de código aberto, com suporte a diversas linguagens e totalmente configurável. O VSCode (https://code.visualstudio.com/) está disponível para Windows, Linux e MacOS, vem com o básico para começarmos a programar e permite a instalação de extensões disponibilizadas pela comunidade para atender uma grande variedade de necessidades.

Sua instalação também é bem simples, bastando executar a instalação padrão, ou baixar o pacote .zip e executar sem a necessidade de instalar. Ao abri-lo, a tela inicial será parecida com a Figura 1.4.

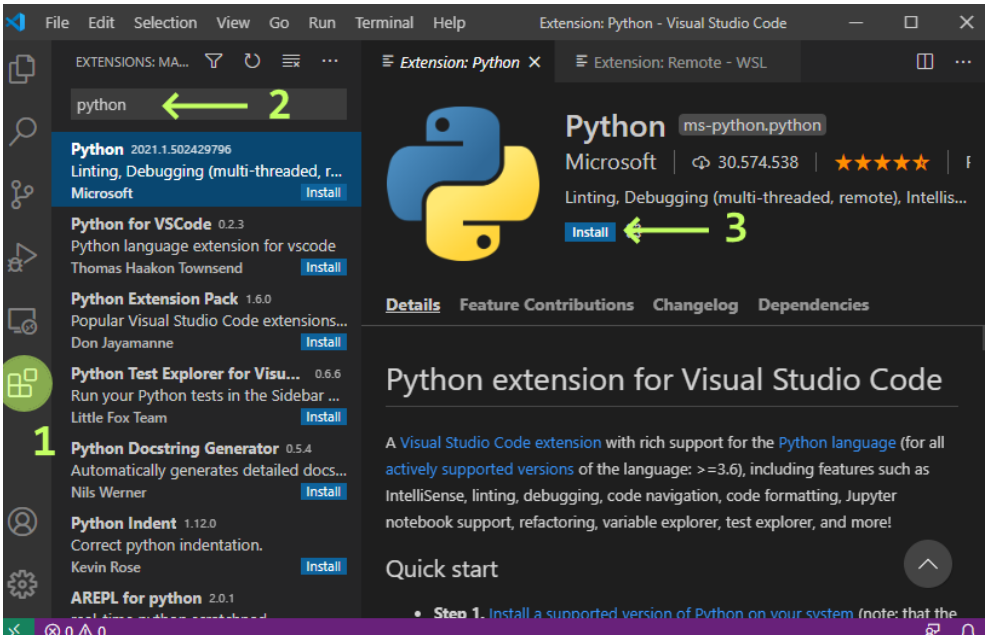
Figura 1.4: Tela inicial do Visual Studio Code



Fonte: do autor, 2021

O VSCode é uma IDE de propósito geral, ou seja, pode ser utilizado para diversas linguagens simultaneamente. Ele provê grande parte das facilidades mencionadas, além de ser leve e ter uma ótima integração com a linguagem Python. No entanto, a instalação padrão vem somente com as funcionalidades básicas, e as ferramentas específicas para cada linguagem devem ser instaladas como extensões, usando a própria interface do VSCode. Para esta disciplina será utilizada a extensão do Python, desenvolvida pela própria Microsoft. Para instalá-la, vá para a aba de extensões na esquerda, busque pela palavra “python” e instale a extensão mostrada na Figura 1.5.

Figura 1.5: Instalação da extensão do Python no VSCode



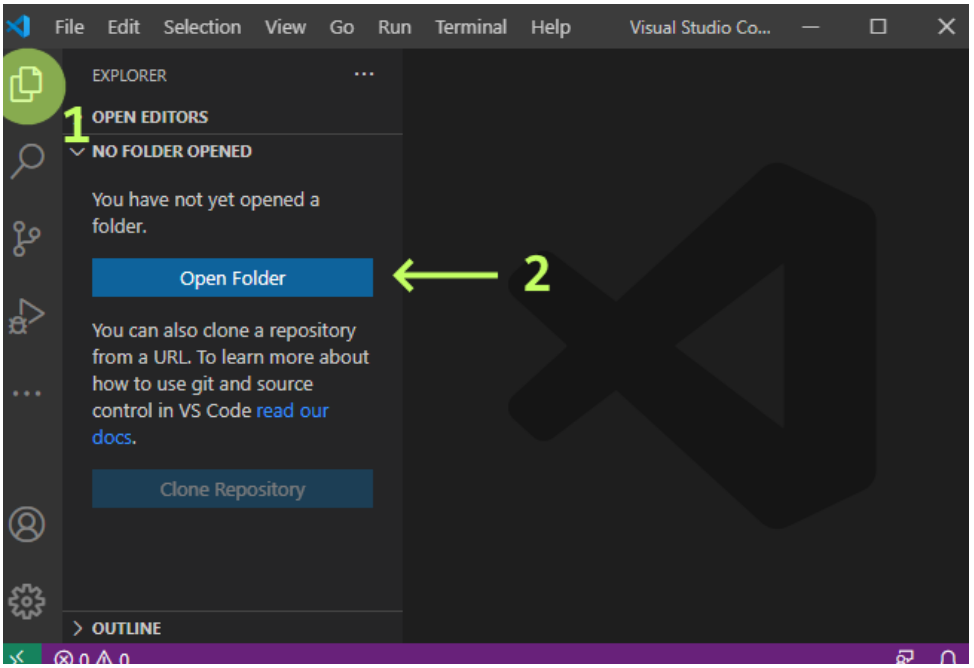
Fonte: do autor, 2021

Instale apenas uma extensão para a linguagem Python, pois ao instalar mais de uma extensão, como a segunda ou a terceira da lista mostrada na figura, elas podem entrar em conflito, produzindo efeitos inesperados e provavelmente indesejados. Com o tempo você aprenderá sobre outras extensões para melhorar sua produtividade, mas como esse é nosso primeiro contato com a IDE, vamos começar usando apenas esta.

Agora estamos quase prontos para executar nosso programa "ola.py" no VSCode, para isso precisamos escolher o projeto que vamos trabalhar. Isso é feito na aba de diretórios, clicando em "*Open folder*"[2], como mostra a Figura 1.6, e selecionando a pasta com os arquivos do projeto que vamos trabalhar. Essa pasta pode estar em qualquer caminho no seu computador, no exemplo da Figura 1.7, selecionamos a pasta "aula01", que contém o arquivo "ola.py" criado anteriormente no bloco de notas.

[2] O VSCode vem configurado em inglês por padrão, para alterar para português, instale a extensão: "*Portuguese (Brazil) Language Pack for Visual Studio Code*", fornecida pela Microsoft, seguindo o mesmo procedimento que fizemos para instalar a extensão do Python.

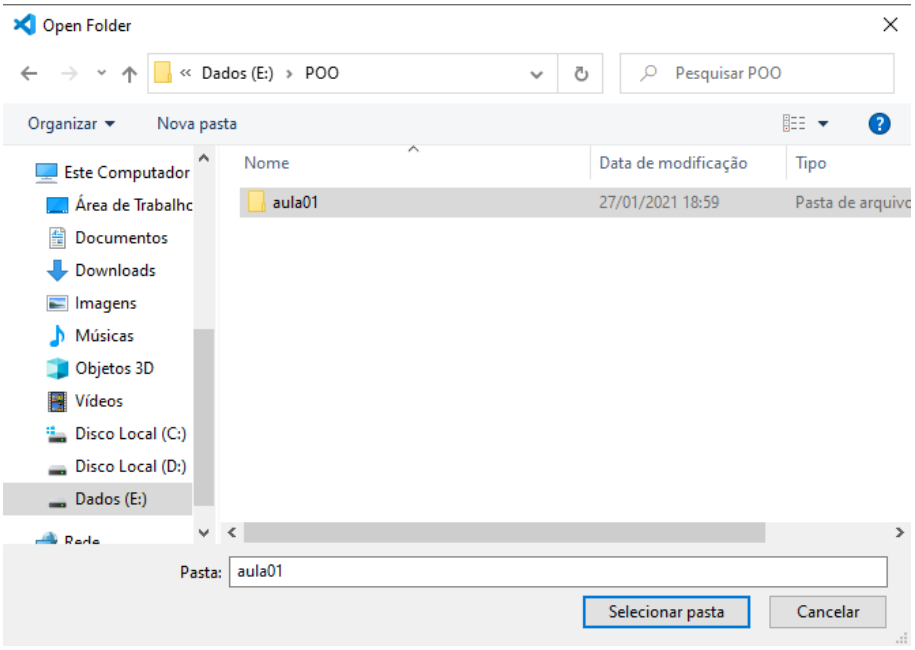
Figura 1.6: Botão para selecionar uma nova pasta de projeto



Fonte: do autor, 2021

Dica de organização: crie uma pasta com o nome da disciplina, por exemplo POO, e crie dentro dela uma pasta para cada aula, conforme avançamos na disciplina. No VSCode, você pode escolher por abrir a pasta da aula em questão, como estamos fazendo aqui, ou abrir a pasta da disciplina, como veremos em outras situações.

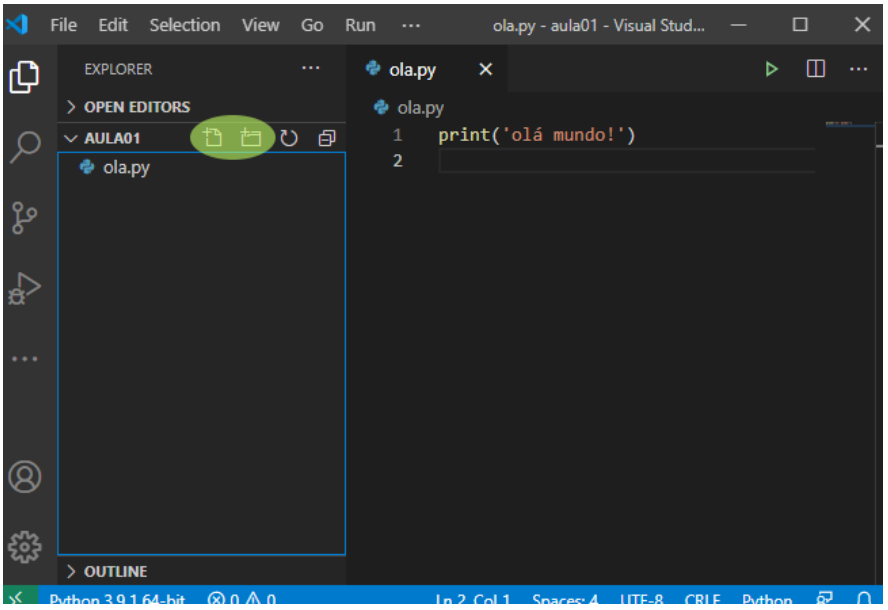
Figura 1.7: Seleção da pasta do projeto no VS Code



Fonte: do autor, 2021

Após selecionar a pasta, ela irá aparecer na parte esquerda da IDE, com algumas opções no topo, como podemos ver na Figura 1.8, para criar novos arquivos e pastas dentro do projeto. Por hora não precisamos criar nada pois nosso projeto já contém o arquivo que criamos com o bloco de notas, mas encorajamos que testem a criação de pastas e arquivos para se familiarizar com seu uso.

Figura 1.8: Visualização da pasta do projeto aberta no VSCode e do arquivo "ola.py" aberto no editor



Fonte: do autor, 2021

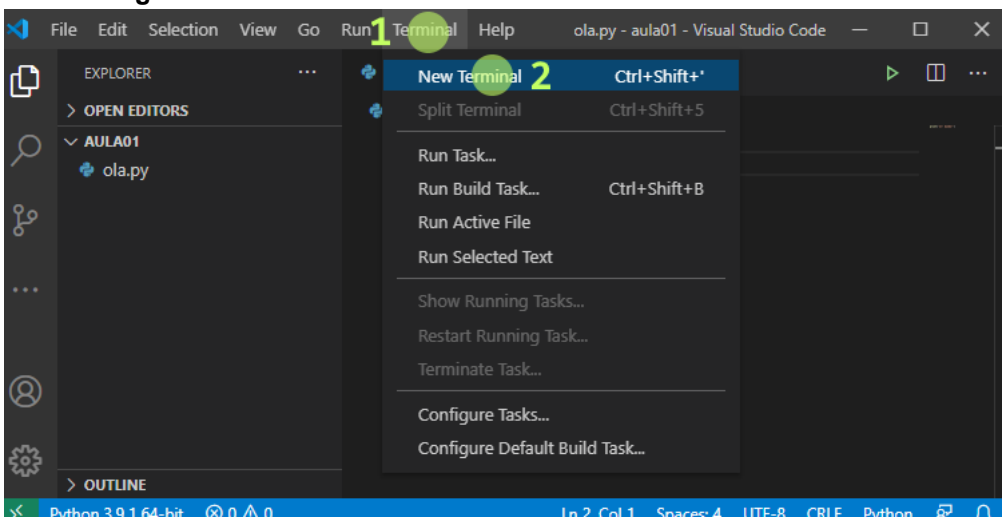
Ao clicar no arquivo “ola.py” ele será mostrado no editor de texto à direita, onde podemos editá-lo. Para salvar o arquivo podemos usar o atalho de teclado “Ctrl + s”. Para executar o arquivo existem duas alternativas:

1) Executá-lo no terminal (equivalente ao prompt de comando), o que pode ser feito manualmente, a partir do menu de contexto ou clicando no ícone de “play” no canto superior direito do editor de texto;
2) Executá-lo no modo “debug”, no qual podemos introduzir pontos de parada ao longo do código e usá-los para controlar a execução e inspecionar as variáveis.

Se você estiver acostumado a programar usando a *Shell* do IDLE, usada em LP, pressionar a tecla F5 irá executar a segunda opção, rodando o código no modo “debug”. Neste momento veremos a opção 1, então para fazer com que o VSCode execute nosso código no terminal, precisamos primeiro abrir um terminal e em seguida executar o comando para rodar nosso arquivo “*.py”.

Seria possível abrir um terminal ou prompt de comando externo, navegar até a pasta do arquivo e digitar o comando para executá-lo, mas lembre-se que o conceito principal de uma IDE é justamente integrar o ambiente de desenvolvimento para que possamos programar e testar em um único lugar. Portanto, para abrir um terminal no próprio VSCode, clique no menu “Terminal” e em seguida em “New Terminal”, ou use o atalho listado a direita do comando no menu (pode variar em função do sistema operacional), como mostra a Figura 1.9.

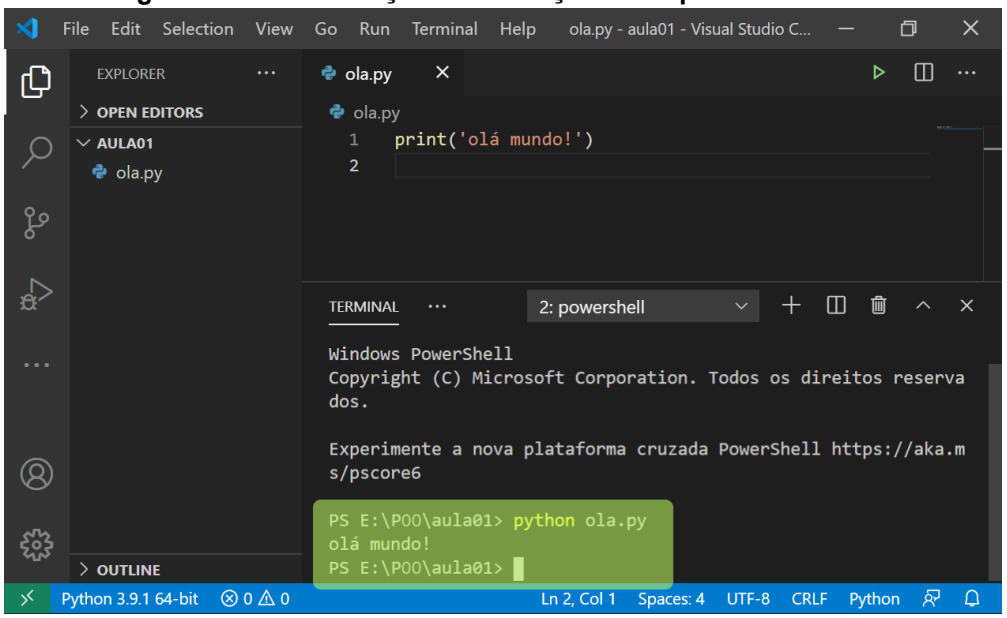
Figura 1.9: Menu de abertura de um novo terminal no VSCode



Fonte: do autor, 2021

O novo terminal será aberto na metade inferior de onde estava o editor de texto. É possível editar qual tipo de terminal se deseja usar, mas por hora podemos ficar com o terminal padrão que o VSCode abrir. No exemplo da Figura 1.10, foi aberto um terminal do Windows PowerShell, mas de maneira geral, os demais são equivalentes.

Figura 1.10: Visualização da execução do arquivo no terminal



Fonte: do autor, 2021

Para executar o arquivo "ola.py" utilizamos o seguinte comando no terminal:

```
> python ola.py
```

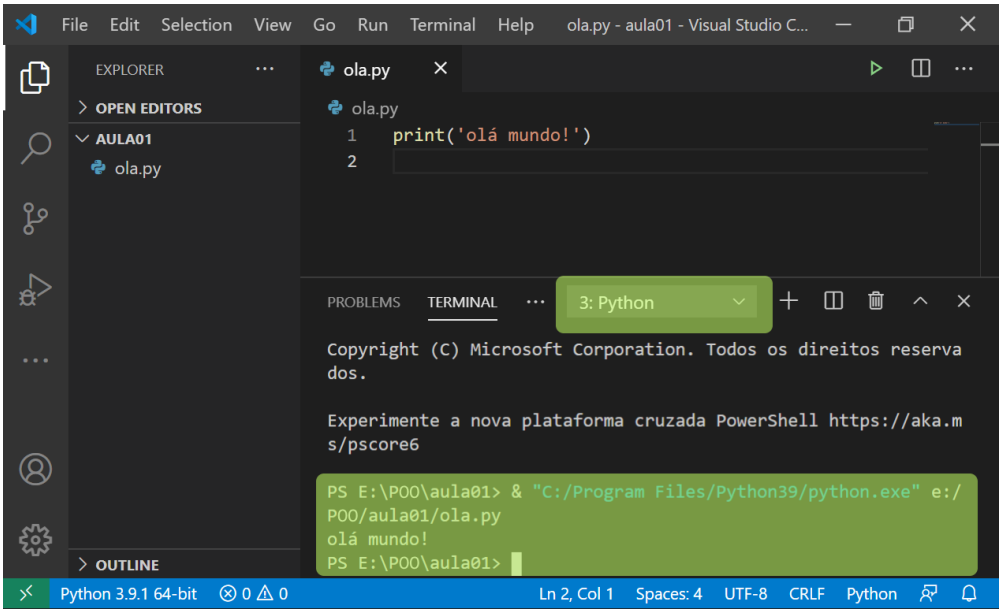
Observe o uso de apenas um sinal `>` para indicar que estamos utilizando um terminal, em contraste com `>>>` presente na *Shell* do Python. Neste comando temos duas partes:

1) `python`: é o comando que o terminal irá executar, no caso o termo *python* é um atalho para o executável do python que está instalado no seu computador.
2) `ola.py`: é passado como argumento para o comando `python`, que neste caso será interpretado como um nome de arquivo. Como não foi dito em que pasta este arquivo está, o Python irá procurá-lo na pasta local, retornando um erro caso o arquivo não seja encontrado.

Observe que antes do sinal `>` no terminal há um caminho para uma pasta, esta é a pasta de onde o terminal está sendo executado. Por padrão, o VSCode sempre abre um novo terminal a partir da pasta do projeto.

Alternativamente, é possível clicar com o botão direito na área do editor de texto do arquivo aberto e selecionar a opção "*Run Python File in Terminal*", o que irá abrir um novo terminal e executar o comando para "rodar" o arquivo com o python. Observe na Figura 1.11 o resultado desse procedimento, e veja que o número do terminal agora é 3, uma vez que antes haviam dois terminais abertos (confira na Figura 1.10). Você pode alternar entre os terminais se quiser.

Figura 1.11: Resultado da execução do arquivo no terminal através do menu de contexto do editor de texto.



Fonte: do autor, 2021

Com isso terminamos a configuração básica do VSCode e estamos prontos para começar a programar na nova IDE. O comando `> python` pode ter um nome ligeiramente diferente dependendo do sistema operacional:

```
> py ola.py                    # Windows
> python3 ola.py               # Linux e MacOS
```

Se o seu computador não reconhece nenhum dos comandos apresentados, isso pode indicar um problema com a instalação do Python no seu computador, certifique-se de que o Python tenha sido instalado para todos os usuários e adicionado ao PATH do seu sistema operacional. Em caso de dúvida, a abordagem mais fácil no Windows é re-instalar o Python, tomando o cuidado de não fazer a instalação padrão; escolha a instalação personalizada e confira as opções mencionadas acima.

1.4. Princípios norteadores e um guia de estilo para Python

Python é uma linguagem de código aberto, e como tal, alterações e melhorias na linguagem são propostas[3], discutidas, avaliadas e implementadas pela própria comunidade de desenvolvedores. Para gerenciar todo esse processo, existe a *Python Software Foundation*, organização sem fins lucrativos responsável por manter e avançar o desenvolvimento da linguagem, veja mais em https://www.python.org/psf-landing/.

Uma das principais ideias que guiaram o criador da linguagem Python, Guido van Rossum, é que programas são lidos com muito mais frequência do que são escritos (Python Software Foundation, 2021).

[3] Esse processo é realizado através das PEP's, ou Propostas de Melhoria do Python (do inglês *Python Enhancement Proposal*). Veja mais em https://www.python.org/dev/peps/.

Pense em uma aplicação qualquer, um trecho de código é escrito apenas uma vez, mas conforme a aplicação evolui, eventualmente precisaremos revisitá-lo, seja para corrigir um bug, para aprimorar o algoritmo utilizado ou para adicionar uma nova funcionalidade às já existentes, e para fazer isso corretamente, é imprescindível entender o que o código que já está lá faz, antes de podermos incluir as modificações necessárias.

Agora imagine que você precise editar um código escrito por outro desenvolvedor, que há anos não trabalha mais na empresa, e que já teve alterações feitas por diversas outras pessoas. Se cada um programar da forma que bem entender, muito rapidamente a situação fica insustentável e você gastaria muito mais tempo apenas tentando entender o que já está ali do que de fato implementando algo novo.

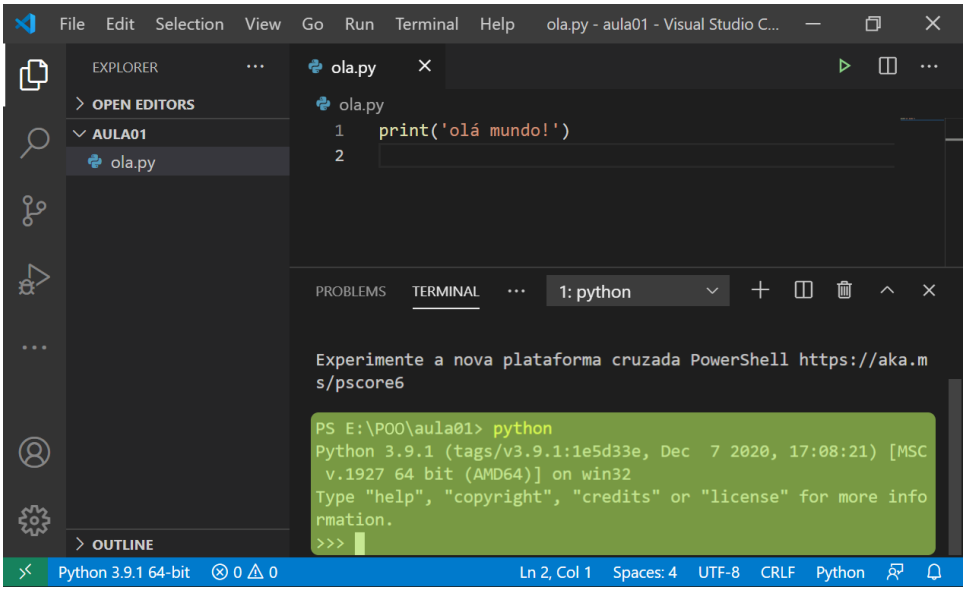
Por isso podemos dizer que a legibilidade e consistência do código são extremamente importantes, tão importantes quanto o algoritmo em si. Para isso, temos um guia de estilo desenvolvido pela comunidade Python e apresentado na proposta de melhoria número 8, a PEP 8 (Rossum, G. V., Warsaw, B., Coghlan, N., 2013).

Tim Peters (1999), resumiu os princípios que guiaram o desenvolvimento da linguagem Python, totalizando 19 aforismas, em um poema que leva o nome de “Zen do Python”. Estes princípios foram incluídos oficialmente na PEP 20, em 2004 (Peter, T., 2004) e podem ser acessados na *Shell* do Python, com o seguinte comando:

```
>>> import this
```

Como estamos trabalhando com o VSCode agora, não seria legal precisarmos voltar ao IDLE sempre que precisarmos testar algo diretamente na *Shell*. Felizmente, podemos acessar a *Shell* de qualquer terminal em um computador que possua o Python instalado, basta digitar o comando *python* no terminal, sem nenhum argumento, como podemos ver na Figura 1.12. Para sair da *Shell* e voltar ao terminal, use o comando `exit()`.

Figura 1.12: Abertura da *Shell* do Python no terminal



Fonte: do autor, 2021

1.4.1. PEP 8 - Guia de Estilo

O guia de estilo para o Python é bastante detalhado e foi desenvolvido pensando justamente em aprimorar tanto a legibilidade quanto a consistência do código. Ele cobre os principais pontos, de como nomear variáveis e funções a como quebrar linhas que ficam muito compridas, passando por quando é recomendado pular linhas no código e como devemos usar variáveis booleanas em estruturas condicionais, entre muitas outras recomendações.

O guia pode ser lido na íntegra em https://www.python.org/dev/peps/pep-0008/, mas não é algo que deve ser lido uma vez e decorado. Pelo contrário, deve ser incorporado organicamente como uma filosofia de desenvolvimento, sendo consultado sempre que necessário.

Como todo guia ou recomendação, isso não diz respeito às regras sintáticas do Python, portanto o fato de um trecho de código não segui-lo não surtirá nenhum efeito do ponto de vista de execução desse código, mas pode reduzir bastante sua legibilidade por outras pessoas que precisarem editá-lo (incluindo você mesmo no futuro).

Conforme avançamos na disciplina, falaremos um pouco mais sobre uma ou outra recomendação, de modo que vocês possam aprendê-las aos poucos, aplicando-as de maneira orgânica em seus códigos de acordo com a necessidade. Outra coisa muito importante quando se trata de um guia de estilo geral como este, é saber quando não se deve usá-lo. Não existe uma resposta correta para essa questão, portanto sempre que estiver em dúvida, não hesite em perguntar e discutir com colegas a melhor abordagem (use o "Zen do Python"[4] para guiar a discussão).

Como dito anteriormente, o guia de estilo foi pensado para melhorar a consistência, mas não adianta ser consistente com o guia se isso faz o código inconsistente com outra parte do código. É comum projetos ou empresas terem o seu próprio guia de estilo, que pode ser ou não baseado no guia geral, portanto é importante saber quando seguir e quando ignorar uma recomendação.

> Um guia de estilo fala sobre consistência. Consistência com este [PEP 8] guia é importante. Consistência interna em um projeto é mais importante. Consistência interna de um módulo ou função é ainda mais importante. (Rossum, G. V., Warsaw, B., Coghlan, N., 2013)

A PEP8 lista também algumas razões e situações em que o mais sensato é ignorar a recomendação, como por exemplo:

[4] Você pode encontrar uma tradução para o poema "*The Zen of Python*" na Wikipédia, em: https://pt.wikipedia.org/wiki/Zen_of_Python.

1) Se por algum motivo aplicar a recomendação torna o código menos legível, incluindo para alguém já familiarizado com o guia da PEP8;
2) Para manter a consistência com o código ao redor, que também quebra determinada recomendação (talvez por motivos históricos, embora esta possa ser uma boa oportunidade de "limpar" a bagunça de outra pessoa);
3) Porque o código em questão foi escrito antes do lançamento da recomendação e não há nenhuma outra razão para modificá-lo.
4) Quando o código em questão precisa ser compatível com uma versão mais antiga do Python que não suporta o recurso utilizado pela recomendação.

Referências

PETERS, T. **The python way**. 1999. Disponível em: <https://mail.python.org/pipermail/python-list/1999-June/001951.html>. Acesso em: 29 jan. 2021.

PETERS, T., **PEP 20** - the zen of python. 2004. Disponível em: <https://www.python.org/dev/peps/pep-0020/>. Acesso em: 27 jan. 2021.

ROSSUM, G. V., **Foreword for "programming python"**. 1996. Disponível em: <https://www.python.org/doc/essays/foreword/>. Acesso em: 27 jan. 2021.

ROSSUM, G. V., WARSAW, B., COGHLAN, N., **PEP 8** - style guide for python code. 2013. Disponível em: <https://www.python.org/dev/peps/pep-0008/>. Acesso em: 27 jan. 2021.